Arena Ticket Booking System

By Leander Almonte and Luke Nwantoly

Feature Selection

How we chose our features?

-Basic Ticket System Features

-Examples: Booking a ticket, Searching for a ticket, Refunding.

-To implement Singleton, TicketSystem class.

-To implement AbstractFactory, we used different types of Events.

-Complexity of the project -> Excluded some features like the receipt class. Did not implement event filters/ticket filters to find a specific ticket.

Technology Choices

- IntelliJ
- SQLite
- JavaFX
- FXML
- SceneBuilder

Work Division

- Luke: Implementing controller methods like Booking a Ticket, Searching for a Ticket, Displaying the Data, Core classes, Unit Testing. Mostly Logic (Controller)

- Leander: Implementing the Database, Core classes, Design Patterns, Designing the GUIs, Internationalization. Mostly Output (View)

Challenges Faced

- Internationalization: We had issues with figuring out how to implement this feature. The main issue was the program wasn't properly detecting the resource bundle
- Unit testing: The unit testing was somewhat simple to do however it required the use of multiple functions to allow is to make sure the methods were functioning as designed
- Time: The end of the semester bombarded us with work, which gave us a tough time trying to find time to work on the project.

Unexpected Ease

- Implementing the Database: Never worked with a Database in a Java project, and we thought it would be complicated to keep the data consistent throughout the project. The Singleton pattern made it easy to load data from the database.
- SceneBuilder: First time working with SceneBuilder, we had doubts about it but when we started using it, it was very easy to learn.

Regrets

- Time Management: We didn't manage our time very well when developing this project which ultimately led us to setting aside a lot of our ideas for this project. We had a lot of ideas for features at the start. However, we really underestimated how much implementing our ideas would take.
- Having too many ideas: Having too many ideas gave us so much confusion on how to proceed with our implementation. When we wanted to implement one part, it would require another part which ultimately led us to a lot of just confusion.

Git Conflicts

- Wrong type of project: We began with a Maven Java project to implement our project with the SQLite dependency and the JavaFX dependency, but whenever we would push the code, the code wouldn't work for one but it would work for the other. We realized that we just had used the wrong type of project, we were supposed to create a JavaFX Application.
- Learning to merge: IntelliJ made it very easy to organize merges and ensure our code is stays functional

Learning Outcomes

- **Implement Design Patterns and understanding their use:** The implementation of design patterns helped us further understand their purpose. The design patterns are what really helped build the foundation of this project.
- **Complexity of using a Database in a project:** The implementation of a database in such a project really made us realize how complicated things can get with data manipulation. It made us realize how important but also complicated it is to ensure the data stays consistent throughout the whole project.

Demo